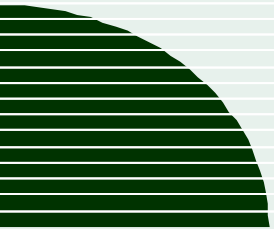
Progetto fisico e tuning del DB

Tecnologie e Sistemi per la Gestione di Basi di Dati e Big Data M



Progetto fisico e tuning

- La valutazione del progetto di un DB avviene tramite la misurazione delle prestazioni del DBMS rispetto alle interrogazioni e alle operazioni di modifica del DB tipicamente utilizzate

- È possibile migliorare le prestazioni
  - A-priori tramite la progettazione (concettuale, logica, fisica)
  - A-posteriori tramite il tuning di parametri e/o oggetti del DB

Le principali problematiche

- Selezione degli indici
  - Su quali attributi conviene costruire un indice?
    - Un indice può velocizzare alcune query, ma tipicamente rallenta le transazioni
- Data clustering
  - Conviene mantenere i dati ordinati? Su quale attributo?
- Data partitioning
  - Conviene partizionare i dati?
    - Fondamentale in ambito distribuito, ma importante anche in DBMS centralizzati
- Modifiche dello schema logico per migliorare le prestazioni
  - Denormalizzazione, replicazione
- Riscrittura di query/transazioni
  - SQL server-side (trigger, stored procedures)

Workload

- La chiave per un buon progetto fisico è ottenere una descrizione (il più possibile) accurata del carico di lavoro atteso
- Il workload include:
  - Un elenco di query con la rispettiva frequenza di esecuzione
  - Un elenco di transazioni (operazioni di modifica) con la rispettiva frequenza di esecuzione
  - Un obiettivo di prestazioni per ciascun tipo di query/operazione di modifica
- La specifica di un workload richiede inoltre una stima delle selettività dei predicati, cosa che in fase di progettazione è più complesso da ottenere…
- Diverso se il workload viene osservato mentre il DBMS è in esecuzione…

DB2 Workload Manager (WLM)

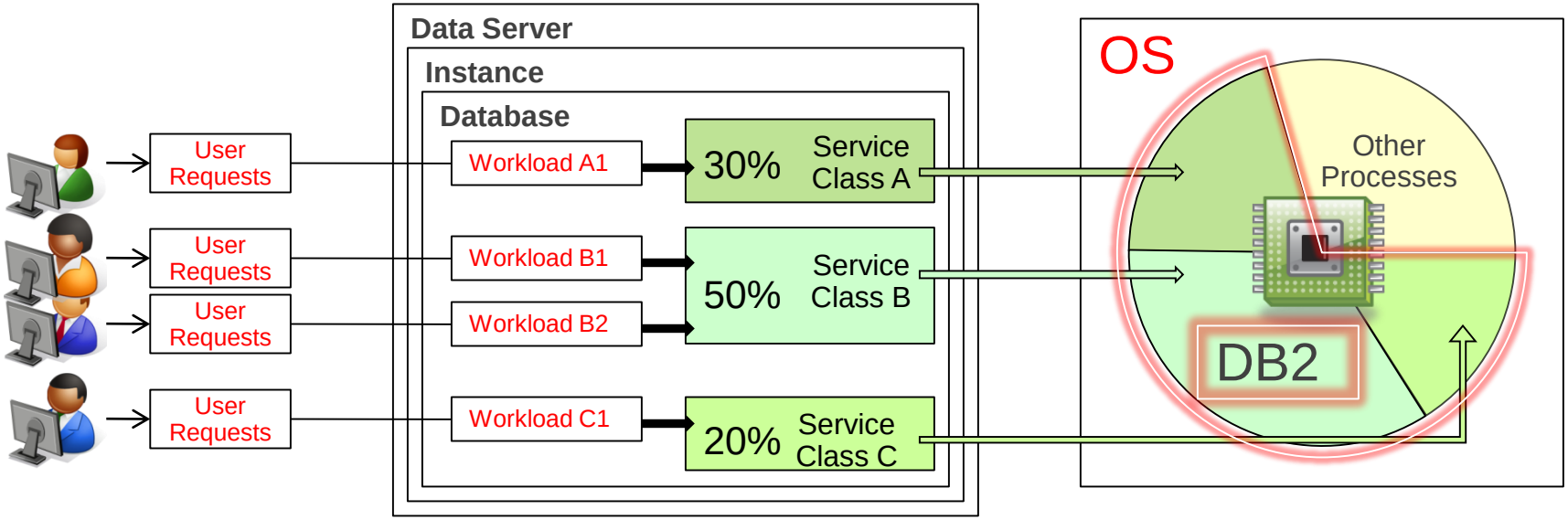
- Strumento che permette di assegnare a diverse applicazioni diverse classi di priorità di esecuzione e utilizzo delle risorse, e di monitorare query e transazioni. I principali "ingredienti" sono:
  - Classi di servizio: per l'allocazione di risorse e la specifica di relative soglie di utilizzo
  - Workloads: insieme di attività legate a un utente/gruppo/IP/… Esistono 2 workload di default:
  - Monitor: per la registrazione di vari tipi di eventi:
    - Attività, locking, …

```
SELECT WORKLOADNAME, SERVICECLASSNAME
FROM   SYSCAT.WORKLOADS;

WORKLOADNAME              SERVICECLASSNAME
--------------------------------------------
SYSDEFAULTUSERWORKLOAD  SYSDEFAULTSUBCLASS
SYSDEFAULTADMWORKLOAD   SYSDEFAULTSUBCLASS
```

- Il collezionamento di dati può avvenire in forma aggregata oppure puntuale (procedura `wlm_collect_stats`)

DB2 Workload Manager (WLM)

The DB2® workload manager (WLM) dispatcher is a built-in DB2 technology by which you can specifically allocate CPU resources to work that is being executed on a database server. CPU resource entitlements can be controlled by using CPU shares and CPU limit attributes on DB2 WLM user and maintenance service class objects.



Tuning

- Il tuning è un'attività comunque necessaria, anche perché le "condizioni al contorno" possono cambiare:
  - Nuove query/transazioni
  - Modifica delle frequenze delle query/transazioni
  - Upgrade/downgrade dell'hardware
  - Modifica/aggiornamento del DBMS
  - …
- La linea di demarcazione tra design fisico e tuning è sottile, e distinguere nettamente queste due attività non è importante
- Ciò che è importante in entrambi i casi è, viceversa, ricordarsi che le scelte di design/tuning devono essere coerenti con quanto il DBMS farà a run-time
  - Es: E' inutile scegliere di costruire un indice se poi i piani di accesso scelti dall'ottimizzatore non lo utilizzeranno mai!

Strumenti automatici

- Tutti i DBMS più diffusi forniscono strumenti automatici per supportare la progettazione fisica e il tuning del DB
  - DB2 design advisor, SQL Server database tuning advisor, Oracle access advisor, …
  - Tipicamente basati su analisi "what-if"
    - Uso dell'ottimizzatore per valutare l'impatto di una possibile scelta
- Ogni sistema ha poi una varietà di strumenti per la gestione fisica dei dati, l'analisi dei piani di accesso, ecc.
- I suggerimenti forniti dagli advisor si riferiscono al workload fornito
  - Non è possibile trarre considerazioni generali che valgano per workload diversi da quello fornito
- Questo significa che non è mai opportuno "sovra-ottimizzare" un DB
  - Occorre comunque che il DBA conosca quali siano le conseguenze di ciascuna azione suggerita dall'advisor

I 5 principi fondamentali

- "Think globally, fix locally"
  - Non limitarsi all'ottimizzazione di singole query
- "Partitioning breaks bottlenecks"
  - Partizionare nel tempo, nello spazio o in risorse
- "Start-up costs are high, running costs are low"
  - Es.: letture sequenziali, compilazione di query, connessione al DB da applicazioni…
- "Render unto server what is due unto server"
  - Sviluppare le applicazioni per ridurre il carico sui server e l'overhead della comunicazione client-server
- "Be prepared for trade-offs"
  - "You want speed: how much are you willing to pay?"

Selezione degli indici

- La selezione degli indici può essere modellata come un problema di ottimizzazione in cui il costo di ogni query/transazione viene modificato dalla presenza/assenza di un indice
- È però evidente che lo spazio delle possibili soluzioni è enorme
  - Il problema è mostrato essere NP-hard
- Rispetto all'adozione di strumenti (semi-)automatici che richiedono il dettaglio di un workload, è possibile considerare un approccio euristico di tipo incrementale
  - Prima si ragiona su quali indici potrebbero migliorare le prestazioni delle operazioni più importanti
  - Quindi si considera se l'aggiunta di ulteriori indici può migliorare la soluzione
- I due approcci non sono in alternativa. Ad esempio, è possibile usare i suggerimenti forniti da un tool come input per scelte più mirate, come conferma di scelte effettuate, ecc.

DB2 design advisor

- Il design advisor di DB2 è una utility che, a partire da un workload, fornisce suggerimenti per la creazione/rimozione di indici e altro
  - Multi-Dimensional Clustering (MDC) tables, partizionamenti, ecc.
- Una descrizione della prima versione si trova in [VZZ+00]
- Per ogni statement SQL è possibile specificare una frequenza di esecuzione
- E' inoltre possibile stabilire un limite alla quantità di memoria aggiuntiva richiesta dagli indici (default: 20% dimensione del DB)
- Il workload può essere fornito esplicitamente con un file di input, oppure desunto dai workload già catalogati in `SYSCAT.WORKLOADS`
- Fino alla versione 9.7 era disponibile un wizard nel Control Center, attualmente il wizard è presente in Data Studio con licenza a pagamento
- In alternativa, si usa da CLP il comando `db2advis` con le varie opzioni, ad es.:
  - -d <database>
  - -q <schema>
  - -i <input-file>
  - -o <output-file>

DB2 design advisor: esempio (1)

```
-- WORKLOAD DI ESEMPIO CON 5 QUERY. File: GSDB-Workload1.sql
-- Output salvato in GSDB-Workload1-out.sql
--#SET FREQUENCY 100
SELECT    *
FROM      ORDER_DETAILS OD, ORDER_HEADER OH, ORDER_METHOD OM
WHERE     OD.ORDER_NUMBER = OH.ORDER_NUMBER
AND       OM.ORDER_METHOD_CODE = OH.ORDER_METHOD_CODE
AND       OD.PRODUCT_NUMBER < :max_pn;

--#SET FREQUENCY 50
SELECT    IL.INVENTORY_YEAR, IL.PRODUCT_NUMBER, SUM(IL.QUANTITY_SHIPPED) AS TOT_QTY
FROM      INVENTORY_LEVELS IL, PRODUCT P
WHERE     IL.PRODUCT_NUMBER = P.PRODUCT_NUMBER
AND       P.PRODUCTION_COST > 0
GROUP BY  IL.INVENTORY_YEAR, IL.PRODUCT_NUMBER;

--#SET FREQUENCY 20
SELECT    IL.INVENTORY_YEAR, IL.PRODUCT_NUMBER, SUM(IL.QUANTITY_SHIPPED) AS TOT_QTY
FROM      INVENTORY_LEVELS IL, PRODUCT P, PRODUCT_BRAND PB
WHERE     IL.PRODUCT_NUMBER = P.PRODUCT_NUMBER
AND       P.PRODUCT_BRAND_CODE = PB.PRODUCT_BRAND_CODE
AND       P.PRODUCTION_COST > 0
AND       PB.PRODUCT_BRAND_IT = :aBrand
GROUP BY  IL.INVENTORY_YEAR, IL.PRODUCT_NUMBER;
```

DB2 design advisor: esempio (2)

```
--#SET FREQUENCY 50
SELECT    *
FROM      ORDER_HEADER OH LEFT JOIN ORDER_DETAILS OD
ON                (OH.ORDER_NUMBER = OD.ORDER_NUMBER)
AND               (OD.QUANTITY > :qty);

SELECT    OD.ORDER_NUMBER, COUNT(*)
FROM      ORDER_DETAILS OD, ORDER_HEADER OH
WHERE     OD.ORDER_NUMBER = OH.ORDER_NUMBER
AND       OH.RETAILER_NAME = :aRetailer          --es.  'AcquaVerde'
GROUP BY  OD.ORDER_NUMBER;
```

- Da notare la possibilità di specificare parametri anziché costanti (ad es. `:qty` anziché `50`)

DB2 design advisor: esempio (3)

```
db2inst1@diva:~/scripts$  db2advis -d GSDB -q GOSALES -i GSDB-Workload1.sql -o GSDB-Workload1-out.sql
-- LIST OF RECOMMENDED INDEXES
-- ===========================
-- index[1],   10.079MB
   CREATE INDEX "DB2INST1"."IDX2004161454290" ON "GOSALES "."ORDER_DETAILS"
   ("PRODUCT_NUMBER" ASC, "UNIT_SALE_PRICE" ASC, "UNIT_PRICE"
   ASC, "UNIT_COST" ASC, "QUANTITY" ASC, "PROMOTION_CODE"
   ASC, "SHIP_DATE" ASC, "ORDER_DETAIL_CODE" ASC, "ORDER_NUMBER"
   ASC) ALLOW REVERSE SCANS COLLECT SAMPLED DETAILED STATISTICS;
   COMMIT WORK ;
-- index[2],    0.013MB
   CREATE INDEX "DB2INST1"."IDX2004161456160" ON "GOSALES "."PRODUCT"
   ("PRODUCTION_COST" ASC, "PRODUCT_NUMBER" DESC) ALLOW
   REVERSE SCANS COLLECT SAMPLED DETAILED STATISTICS;
   COMMIT WORK ;
….
-- index[8],    0.677MB
   CREATE INDEX "DB2INST1"."IDX2004161500450" ON "GOSALES "."ORDER_DETAILS"
   ("ORDER_NUMBER" ASC) ALLOW REVERSE SCANS COLLECT SAMPLED DETAILED STATISTICS;
   COMMIT WORK ;
```

DB2 design advisor: esempio (4)

```
-- UNUSED EXISTING INDEXES
-- ==========================
-- DROP INDEX "DB2INST1"."IX4_1685581043";
-- DROP INDEX "DB2INST1"."IX5_1685581043";
-- DROP INDEX "DB2INST1"."IX6_1685581043";
-- DROP INDEX "DB2INST1"."IX7_1685581043";
-- ==========================
--
```

- Per ogni indice suggerito è inoltre fornito il "benefit" (in timeron) derivante dalla sua introduzione per i vari statement (uno o più)

Linee guida per la selezione degli indici

- Creazione di un indice
  - Non creare indici inutili
  - Creare prima indici che velocizzino più di una query
  - Considerare l'impatto dei costi di modifica, quindi attenzione a creare indici su attributi soggetti spesso ad aggiornamenti!
- Scelta degli attributi da indicizzare
  - Quelli presenti nelle clausole `WHERE`, `GROUP BY`, `ORDER BY`
- Uso di indici composti
  - Valutare la possibilità di permettere piani index-only
  - Attenzione all'ordine degli attributi in caso di predicati di range
    - Vedi Range-delimiting e Index-SARGable

- Vediamo alcuni esempi e quali siano le considerazioni da fare nei vari casi in termini (più o meno) qualitativi…

Selezione degli indici: esempio (i)

```
SELECT  E.name, D.mgr
FROM    Employees E, Departments D
WHERE   E.dno = D.dno
AND     D.name = 'Toy'
```

- Quali indici?
  - `D.name`
  - `E.dno`?
  - `D.dno`?
- Un possibile (ragionevole) piano di accesso sarebbe:
  - Recuperare i dipartimenti con un indice su `D.name`
  - Usare un index nested loop su E (indice su `E.dno`)
- Quindi, indice (hash) su `E.dno`
- Ai fini della query un indice su `D.dno` non è utile

Selezione degli indici: esempio (ii)

```
SELECT  E.name, D.mgr
FROM    Employees E, Departments D
WHERE   E.dno = D.dno
AND     D.name = 'Toy'
AND     E.age = 25
```

- Se esistesse già l'indice su `E.age` potrebbe essere inutile creare l'indice su `E.dno` (dipende dalle selettività)

Selezione degli indici: esempio (iii)

```
SELECT  E.name, D.mgr
FROM    Employees E, Departments D
WHERE   E.dno = D.dno
AND     E.sal BETWEEN 10000 and 20000
AND     E.hobby = 'stamps'
```

- La scelta cadrà sull'indice più selettivo tra `E.sal` e `E.hobby` con l'aggiunta di `D.dno`
- Se la selettività non fosse nota è ragionevole definire un indice su `E.hobby`
- In alternativa potrebbero essere utili entrambi gli indici facendo intersezione dei RID risultanti

Indici multi-attributo

```
SELECT  E.eid
FROM    Employees E
WHERE   E.age BETWEEN 20 AND 30
AND     E.sal BETWEEN 3000 AND 5000
```

- Un indice (B$^+$-tree) su (`age`, `sal`) potrebbe essere utile
- E un indice su (`sal`, `age`)?
  - Scelte equivalenti se le selettività sono le stesse

- In questo caso:

```
SELECT  E.eid
FROM    Employees E
WHERE   E.sal BETWEEN 3000 AND 5000
AND     E.age = 25
```

- con (`age`, `sal`) entrambi i predicati sono range-delimiting
- con (`sal`, `age`) solo quello su `sal` lo è, l'altro è (solo) index-SARGable
  - quindi più foglie da leggere

Indici per piani index-only (i)

```
SELECT  D.mgr
FROM    Departments D, Employees E
WHERE   D.dno = E.dno
```

- Se abbiamo un indice su `E.dno` possiamo usarlo per un index nested loop senza accedere ai record di `Employees`

```
SELECT  D.mgr, E.eid
FROM    Departments D, Employees E
WHERE   D.dno = E.dno
```

- Il piano non è più index-only (per `Employees`)
- Con un B$^+$-tree su (`dno`, `eid`) il piano torna a essere index-only per `Employees`
  - Un indice hash non andrebbe bene, perché non viene fornito in input nessun valore di chiave per `eid`

Indici per piani index-only (ii)

```
SELECT  E.*, D.mgr
FROM    Departments D, Employees E
WHERE   D.dno = E.dno
```

- Con un B$^+$-tree su (`dno`,`mgr`) è possibile un piano index-only per `Departments` (usata come relazione interna)
- E' anche possibile definire un indice sul solo `dno` che "includa" `mgr`
  - In questo caso i valori di `mgr` non fanno parte delle chiavi di ricerca, ma compaiono solo nelle foglie

```
CREATE UNIQUE INDEX IDX_D_DNO
ON TABLE Departments (dno)
INCLUDE (mgr)
```

- In DB2 l'inclusione di altri attributi è possibile solo per indici `UNIQUE`

Indici per piani index-only (iii)

```
SELECT  E.dno, COUNT(*)
FROM    Employees E
GROUP BY E.dno
```

- Se abbiamo un indice su `E.dno` possiamo usarlo per un piano index-only

```
SELECT  E.dno, COUNT(*)
FROM    Employees E
WHERE   E.sal = 10000
GROUP BY E.dno
```

- È meglio avere un indice su (`sal`, `dno`) o su (`dno`, `sal`)?
  - Meglio (sal,dno), se il predicato è molto selettivo
- Se il predicato fosse `E.sal > 10000` la scelta diventa più complessa da fare, la selettività è evidentemente il fattore determinante

Indici per piani index-only (iv)

```
SELECT  E.dno, MIN(E.sal)
FROM    Employees E
GROUP BY E.dno
```

- È meglio avere un indice su (`sal`, `dno`) o su (`dno`, `sal`)?
  - In questo caso, un B$^+$-tree su (`dno`, `sal`) ci darebbe immediatamente il risultato
  - Il piano che usa l'indice (`sal`, `dno`) richiederebbe anch'esso un B$^+$-tree ma sarebbe comunque meno efficiente

```
SELECT  AVG(E.sal)
FROM    Employees E
WHERE   E.age = 25
AND     E.sal BETWEEN 3000 AND 5000
```

- Un B$^+$-tree su (`age`, `sal`) ci darebbe immediatamente il risultato
- Il piano che usa l'indice (`sal`, `age`) richiederebbe anch'esso un B$^+$-tree ma sarebbe comunque meno efficiente

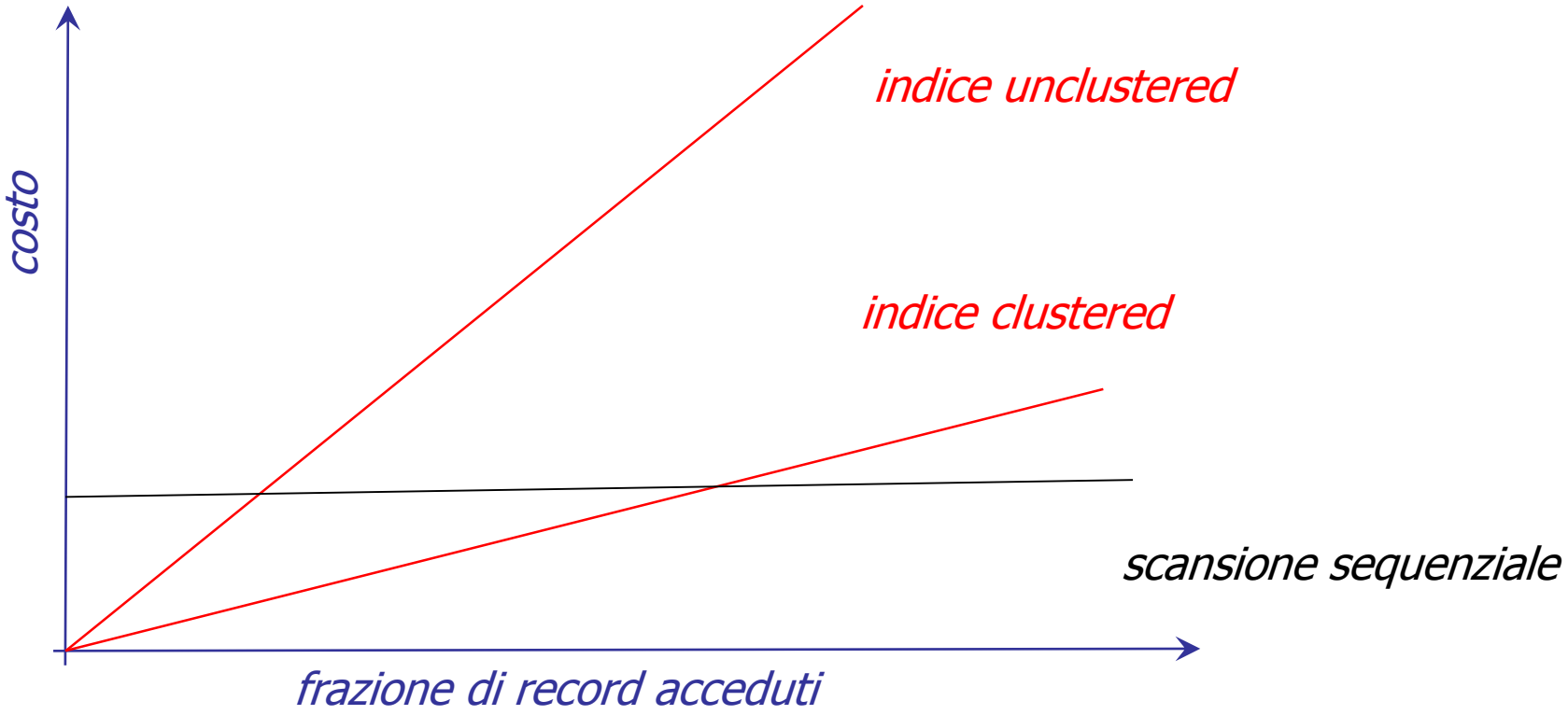
Data clustering

- Il clustering dei dati avviene, come visto, diversamente nei vari DBMS
- In DB2 viene definito un indice che può essere "più o meno" clustered
  - Quando il grado di clustering scende sotto un certo livello è opportuno riorganizzare fisicamente la table (e gli indici associati)
- In Oracle una IOT può essere definita solo sulla primary key
  - In questo caso lo scopo è velocizzare le query su tale chiave
- In SQLServer l'obiettivo è lo stesso, con la flessibilità di poter scegliere qualsiasi clustering key

- In generale:
  - Le query di range sono quelle che beneficiano maggiormente dalla clusterizzazione di un indice
  - Anche le ricerche '=' su attributi non chiave traggono vantaggio da indici clustered in caso di molti duplicati
  - I piani di accesso index-only non necessitano di indici clustered

Indici vs scansione sequenziale

- In generale, il costo di accesso ai record di una tabella per una query con selezione segue l'andamento in figura



Indici clustered: esempio (i)

```
SELECT  E.dno
FROM    Employees E
WHERE   E.age > 40
```

- L'indice su age (B^+-tree, ovviamente) dovrebbe essere clustered?
  - Se ci sono molti ultra-quarantenni l'indice non è utile, quindi dipende dalla selettività

```
SELECT  E.dno, count(*)
FROM    Employees E
WHERE   E.age > 20
GROUP BY E.dno
```

- L'indice su age è utile?
  - Dipende dalla selettività del predicato
- Un'alternativa potrebbe essere un indice su dno
  - In questo caso l'indice dovrebbe essere clustered
  - Con indice unclustered su dno, l'ottimizzatore sceglierebbe di raggruppare E mediante sorting o hashing

Indici clustered: esempio (ii)

```
SELECT  E.dno
FROM    Employees E
WHERE   E.hobby = 'stamps'
```

- L'indice su hobby deve essere clustered?
  - Dipende dalla selettività del predicato su hobby

- E in questo caso? (eid è chiave)

```
SELECT  E.dno
FROM    Employees E
WHERE   E.eid = 552
```

Indici clustered: esempio (iii)

```
SELECT  E.name, D.mgr
FROM    Employees E, Departments D
WHERE   E.dno=D.dno
AND     D.name='Toy'
```

- Gli indici su `D.name` e `E.dno` devono essere clustered?
  - Probabilmente, ci saranno pochi dipartimenti che soddisfano il predicato, quindi indice unclustered
  - Viceversa, l'indice per l'index nested loop dovrebbe essere clustered
    - Per ogni valore di `D.dno` è necessario accedere a molte tuple di `Employees`

Collezionare statistiche in DB2

- Le informazioni statistiche su tabelle e indici sono fondamentali per permettere all'ottimizzatore di operare scelte accurate e per poter ragionare in fase di tuning
- In DB2 si usa il comando `RUNSTATS`
- Esistono molte varianti, tra cui:

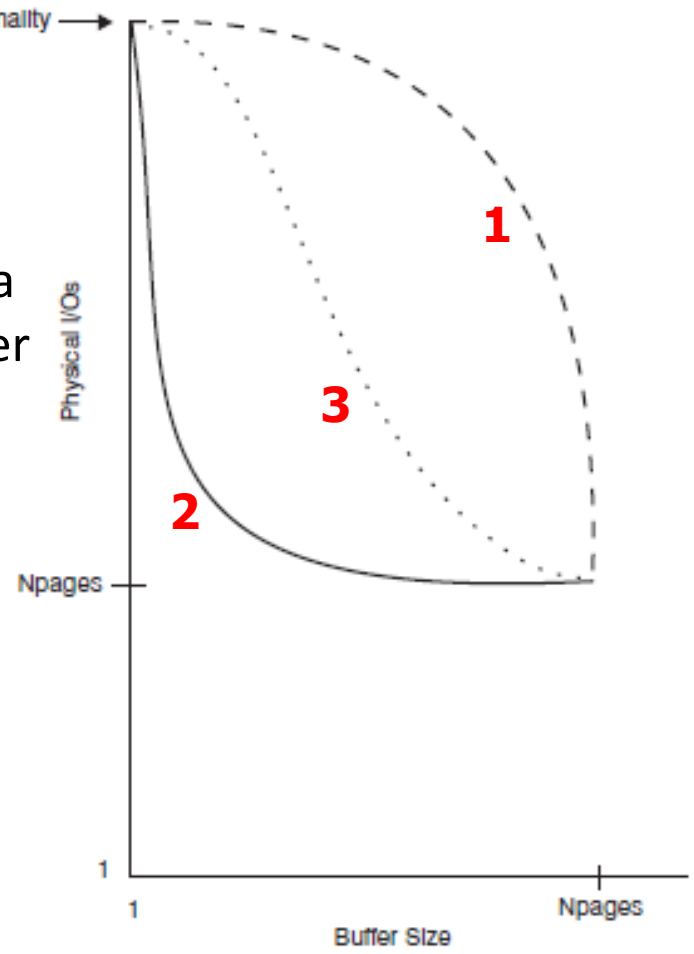
```
RUNSTATS ON TABLE MySchema.TableName
WITH DISTRIBUTION ON ALL COLUMNS
AND DETAILED INDEXES ALL
```

- La forma `WITH DISTRIBUTION` indica di collezionare statistiche dettagliate sulle distribuzioni dei valori (istogrammi e valori frequenti) delle varie colonne (`ON ALL COLUMNS`)
- La forma `DETAILED INDEXES ALL` genera informazioni utili per capire lo "stato di salute" di un indice e il costo di un index scan

Statistiche detailed sugli indici (1)

- In particolare, il sistema calcola un valore di CLUSTERFACTOR e genera una lista di 11 PAGE_FETCH_PAIRS

- Queste ultime sono coppie (buffer_size,#I/O) e stimano quanti I/O (fisici) sarà necessario eseguire per effettuare una scansione completa avendo a disposizione un certo numero di buffer
- Le curve tipiche sono:
  - 1: clustering praticamente nullo
  - 2: buon clustering, avendo un buffer size modesto si evitano I/O inutili
  - 3: situazione intermedia, assunta dall'ottimizzatore in assenza di statistiche dettagliate



Statistiche detailed sugli indici (2)

- Il valore di CLUSTERFACTOR è compreso tra 0 e 1 e rappresenta il grado di correlazione tra i valori di chiave dell'indice e la sequenza fisica dei dati
- Un valore <0.8 suggerisce di riorganizzare i dati se l'indice è supposto essere clustered (-1 indica che il valore non è stato calcolato)

```
SELECT  COLNAMES, CLUSTERFACTOR, PAGE_FETCH_PAIRS
FROM    SYSSTAT.INDEXES
WHERE   TABSCHEMA = 'GOSALES'
AND     TABNAME = 'ORDER_DETAILS' -- ha 2001 pagine
AND     CLUSTERFACTOR != -1;

          COLNAMES     CLUSTERFACTOR PAGE_FETCH_PAIRS
-------------------------------------------------------------------
+ORDER_DETAIL_CODE     1             100 2001 231 2001 … 2001 2001
```

Statistiche detailed sugli indici (3)

- Per un esempio più significativo generiamo dei dati a caso…

```
CREATE TABLE TEST2 (
A       INT,
B       INT,
C       CHAR(250)       ); -- per aumentare la dimensione

INSERT INTO TEST2 VALUES (1,10,' ');

CREATE TRIGGER AUTO_INSERT2
AFTER INSERT ON TEST2
FOR EACH STATEMENT
WHEN    (10000 > (SELECT COUNT(*) FROM TEST2))
INSERT INTO TEST(A,B,C)
        SELECT MOD(3*A+1,20),MOD(3*B+1,100),C FROM TEST2;

INSERT INTO TEST2 VALUES (2,20,' ');
```

Statistiche detailed sugli indici (4)

- Vediamo gli effetti…

```
SELECT A,COUNT(*) AS COUNT
FROM TEST2
GROUP BY A
ORDER BY A; -- 5 valori

A       COUNT
-------------
0        2080
1        2016
2        8192
3        2080
4        2016
```

```
SELECT B,COUNT(*) AS COUNT
FROM TEST2
GROUP BY B
ORDER BY B; -- 28 valori

B       COUNT
-------------
0        1287
1         715
4         286
10          1
...       ...
90       1287
91          1
94         78
```

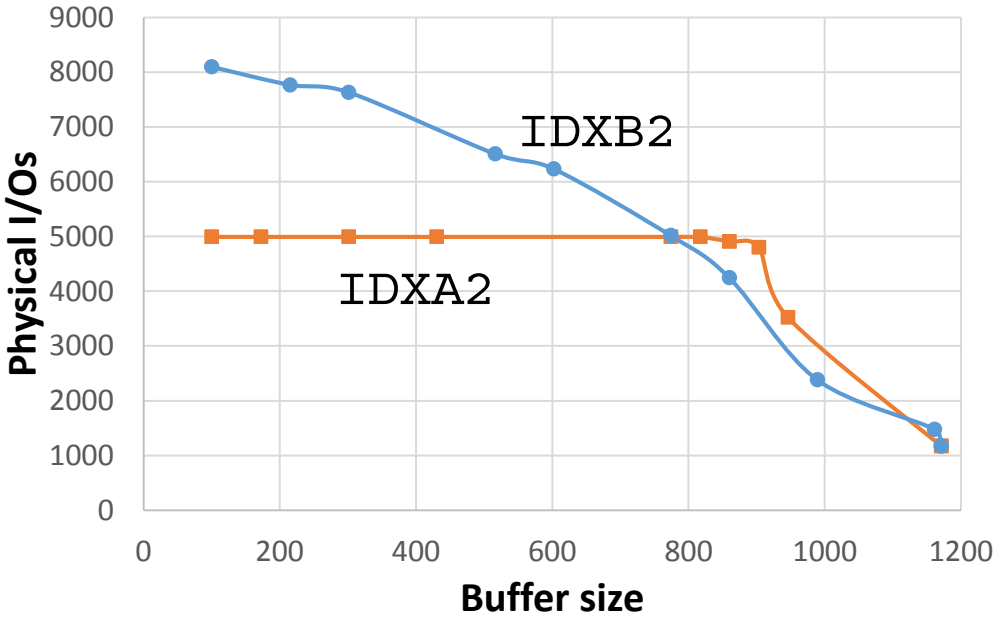
Statistiche detailed sugli indici (5)

- Creiamo gli indici, collezioniamo le statistiche ed esaminiamo i cataloghi…
  - La procedura ADMIN_CMD permette di eseguire comandi DB2 quali RUNSTATS (che non è un'istruzione SQL!) da qualsiasi client, per i quali altrimenti è necessario usare CLP

```
CREATE INDEX IDXA2  ON TEST2(A);
CREATE INDEX IDXB2  ON TEST2(B);
CALL SYSPROC.ADMIN_CMD('RUNSTATS ON TABLE B16884.TEST2
        WITH DISTRIBUTION ON ALL COLUMNS AND DETAILED INDEXES ALL');

SELECT   COLNAMES, CLUSTERFACTOR, PAGE_FETCH_PAIRS
FROM     SYSSTAT.INDEXES
WHERE    TABSCHEMA = 'B16884'
AND      TABNAME = 'TEST2'; -- ha 1171 pagine

COLNAMES CLUSTERFACTOR     PAGE_FETCH_PAIRS
------------------------------------------------------------
+A               0.7486      100 4995 172 4995 301 4995 430 4995
                             774 4995 817 4993 860 4910 903 4803
                             946 3523 1171 1171 1171 1171
+B               0.5392      100 8097 215 7769 301 7634 516 6509
                             602 6235 774 5017 860 4246 989 2384
                            1161 1480 1171 1171 1171 1171
```

Statistiche detailed sugli indici (6)

- Infine osserviamo i grafici corrispondenti alle liste di `PAGE_FETCH_PAIRS`

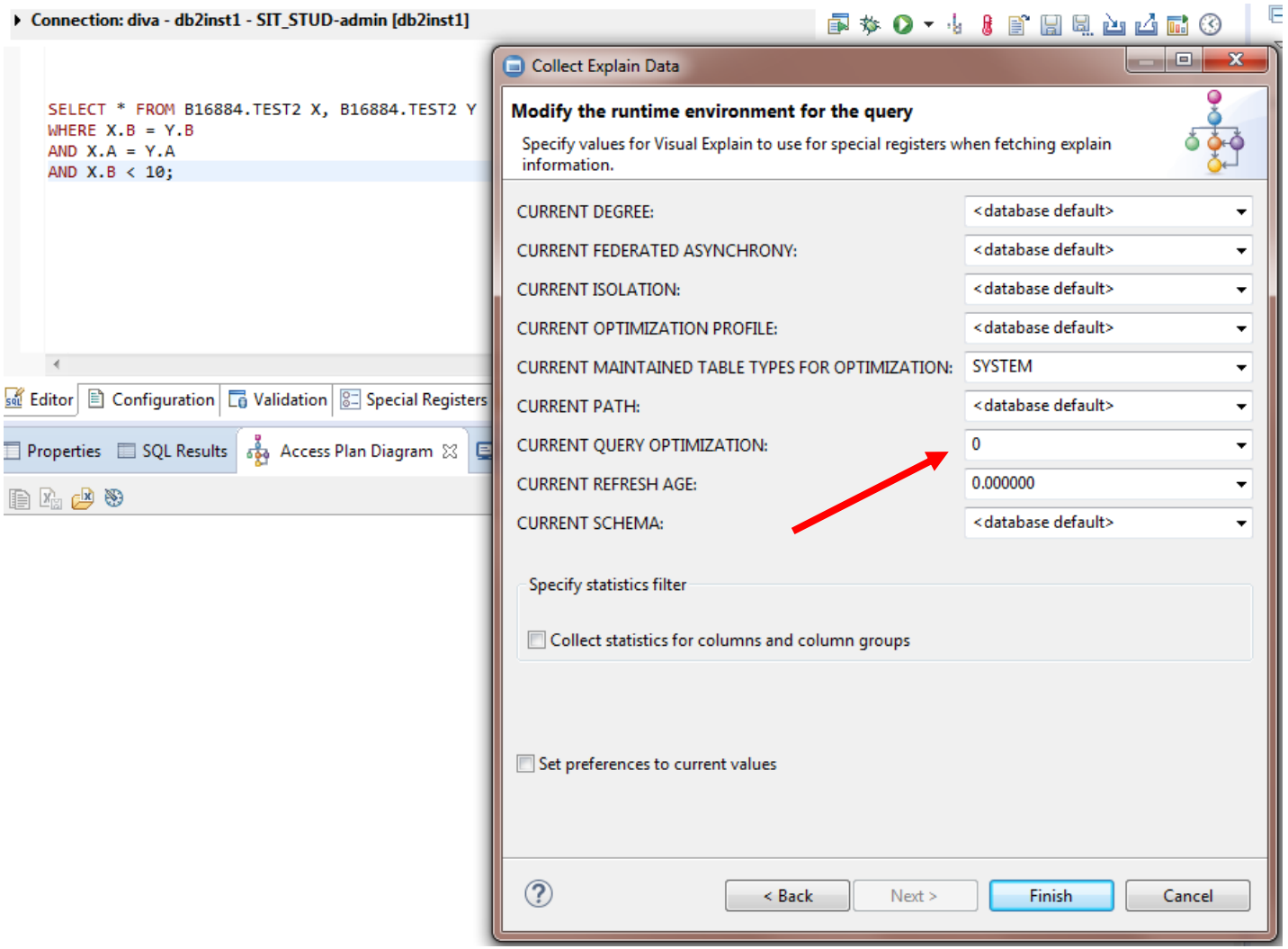


Visualizzazione dei piani di accesso

- Allo scopo di capire meglio come la presenza di indici può influenzare l'esecuzione di una query, è estremamente utile analizzare i dettagli del piano di accesso scelto dall'ottimizzatore
- Un esempio di tool utile allo scopo è il Visual Explain di DB2
  - Prima era parte del Control Center, ora è integrato in Data Studio
- Oltre a fornire informazioni sul piano di accesso nel suo complesso permette di visualizzare le proprietà dei singoli operatori, degli indici e delle table coinvolte
- E' possibile scegliere il livello di ottimizzazione

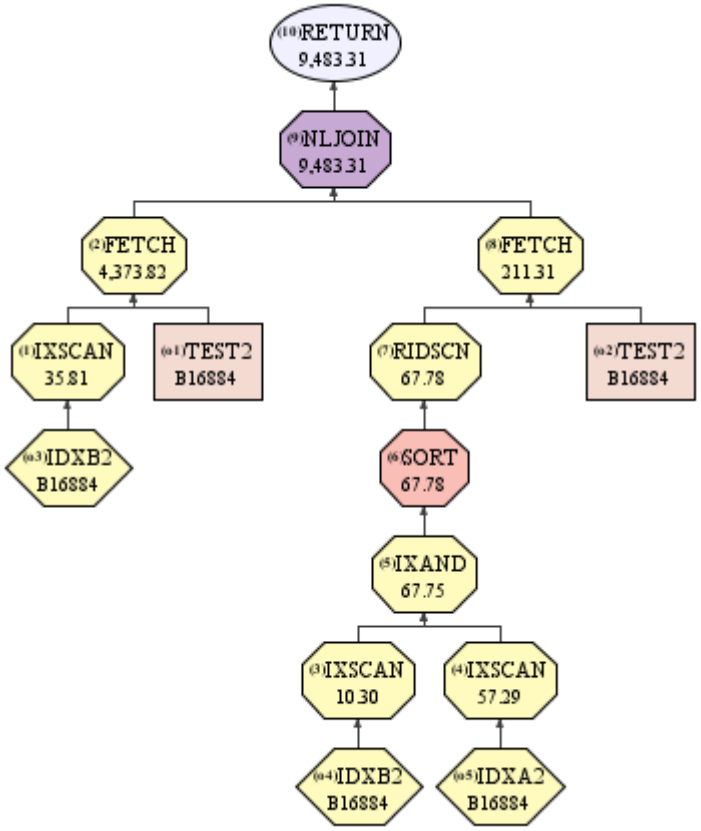
DB2 Visual Explain (1)

- Si consideri la seguente query (sef-join), eseguita a livello di ottimizzazione 0:

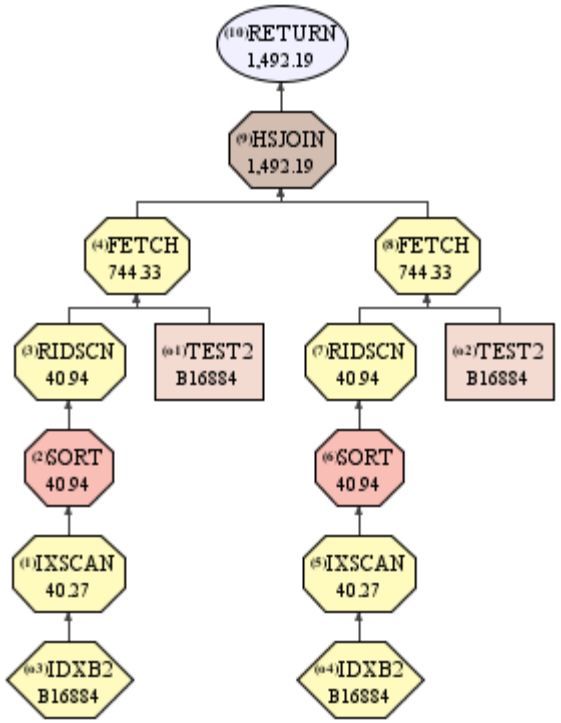


DB2 Visual Explain (2)

- Il relativo access plan:



- … e quello a livello 5:



DB2 Visual Explain (3)

- Si possono visualizzare le proprietà di table…

Table

Accessed table: **B16884.TEST2**:

Open a dialog to view detail

Properties

Name	Value
Name	TEST2
Schema	B16884
Object type	Table (untyped)
Creation time	2020-04-19 17:14:48.408807
Statistics time	2020-04-19 17:16:39.3258
Statistics Information	
Number of columns	3
Number of rows	16384
Width	267
Number of buffer pool pages	1171
Number of data partitions	1
Indicator of distinct row values	N
Tablespace transfer rate	0.06
Statistics Source	Single Node
Prefetch size	32
Extent size	32
Table overflow	0
Percentage compressed	0

Frequent Values

A - [INTEGER]

Sequence	Column Value	Number of Items with Column Value
1	2	8192
2	0	2080
3	3	2080
4	1	2016
5	4	2016

B - [INTEGER]

Sequence	Column Value	Nu Co
1	63	17
2	44	17
3	33	17
4	54	17
5	0	12
6	81	12
7	90	12
8	51	12
9	71	71
10	50	71

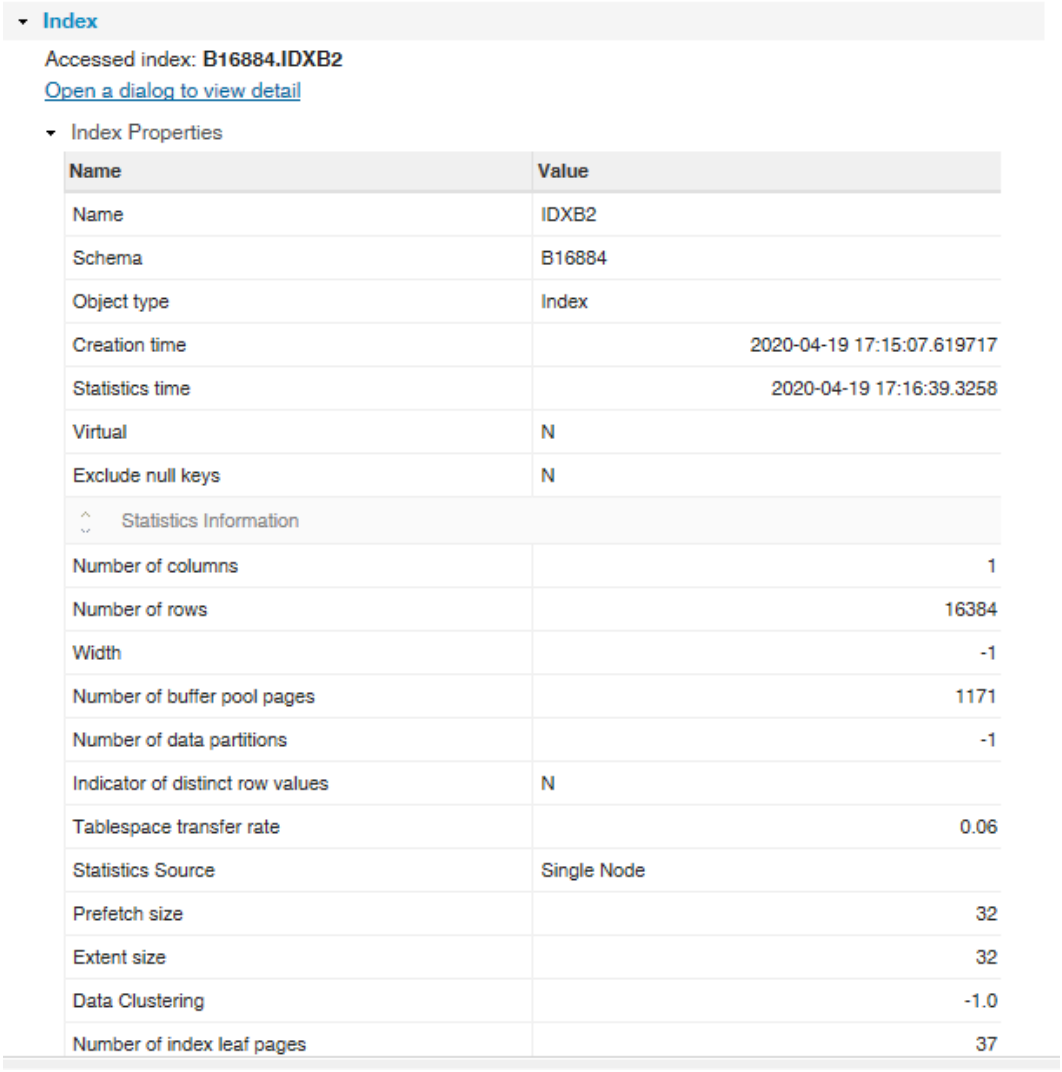
Quantiles

A - [INTEGER]

Sequence	Column Value	Number of Items with Column Value	Number of Distinct Values below Column Value
1	0	0	1
2	0	2080	1
3	1	2080	2
4	1	4096	2
5	2	4096	3
6	2	12288	3
7	3	12288	4
8	3	14368	4

DB2 Visual Explain (4)

- …indici…



Index

Accessed index: **B16884.IDXB2**

Open a dialog to view detail

Index Properties

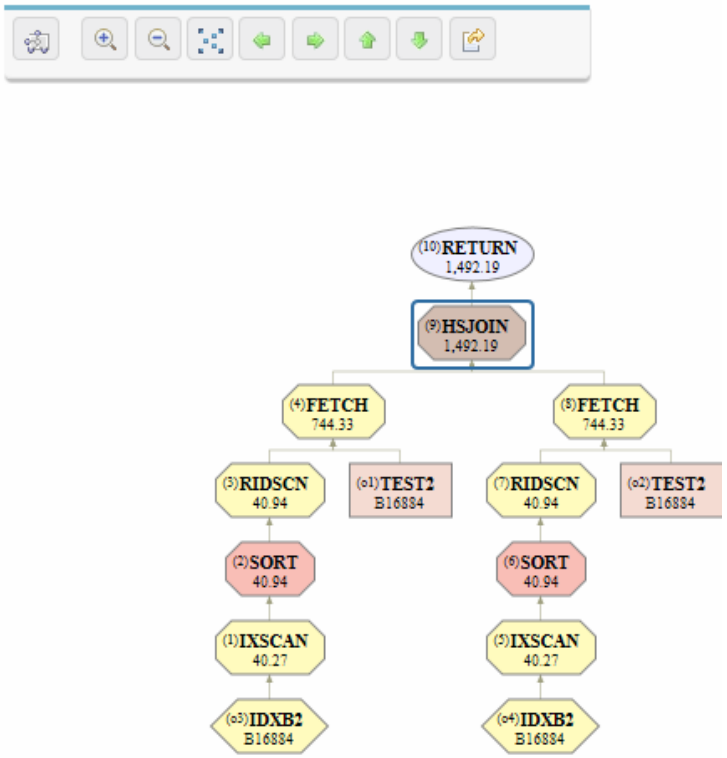
Name	Value
Name	IDXB2
Schema	B16884
Object type	Index
Creation time	2020-04-19 17:15:07.619717
Statistics time	2020-04-19 17:16:39.3258
Virtual	N
Exclude null keys	N
Statistics Information	
Number of columns	1
Number of rows	16384
Width	-1
Number of buffer pool pages	1171
Number of data partitions	-1
Indicator of distinct row values	N
Tablespace transfer rate	0.06
Statistics Source	Single Node
Prefetch size	32
Extent size	32
Data Clustering	-1.0
Number of index leaf pages	37

DB2 Visual Explain (5)

- … e operatori

Properties

Name	Value
Operator Identifier	2
Predicate text	(Q2.A = Q1.A)
Predicate text	(Q2.B = Q1.B)
Operator Type	Hash join
Early Out flag	NONE
Hash Join Bit Filter used	FALSE
Temporary Table Page Size	4096
Hash Code Size	24 BIT
HASHTBSZ	20
TUPBLKSZ	16000
Cost Information	
Estimated Output Cardinality	269,581.844
Cumulative Total Cost	1,492.19
Cumulative CPU Cost	44,021,900.00
Cumulative I/O cost	1,398.85
Cumulative First Row Total Cost	1,492.19
Total cost	3.53
CPU cost	15,218,506.00
I/O cost	0.00
Estimated Bufferpool Buffers	718.64



Riorganizzazione di dati e indici

- Il comando REORGCHK (non supportato da ADMIN_CMD) serve per capire se una table deve essere riorganizzata fisicamente (idem per gli indici)
  - Esegue automaticamente anche RUNSTATS
- Per le table vengono forniti tre indicatori, con relativi valori di soglia
  - F1: % di record in overflow (< 5%)
  - F2: % di spazio utilizzato nelle pagine allocate (> 70%)
  - F3: NPAGES/FPAGES (> 80%)

```
db2 -t
db2 =>
db2 =>
db2 => REORGCHK ON TABLE B16884.TEST2;

RUNSTATS in esecuzione....


Statistiche della tabella:

F1: 100 * OVERFLOW / CARD < 5
F2: 100 * (Utilizzo effettivo di spazio di pagine di dati) > 70
F3: 100 * (Pagine richieste / Numero di pagine totali) > 80

SCHEMA.NAME                     CARD     OV     NP     FP ACTBLK    TSIZE  F1  F2  F3 REORG
-------------------------------------------------------------------------------------------
Tabella: B16884.TEST2
                               16384      0   1171   1171      -  4440064   0  94 100 ---
-------------------------------------------------------------------------------------------
```

Riorganizzazione di dati e indici

- Per riorganizzare i dati e gli indici:

```
REORG TABLE MySchema.MyTable
```

- La forma

```
REORG TABLE MySchema.MyTable INDEX AnIndex
```

  forza la riorganizzazione secondo i valori dell'indice `AnIndex`

- Per riorganizzare solo gli indici:

```
REORG INDEXES ALL FOR TABLE MySchema.MyTable
```

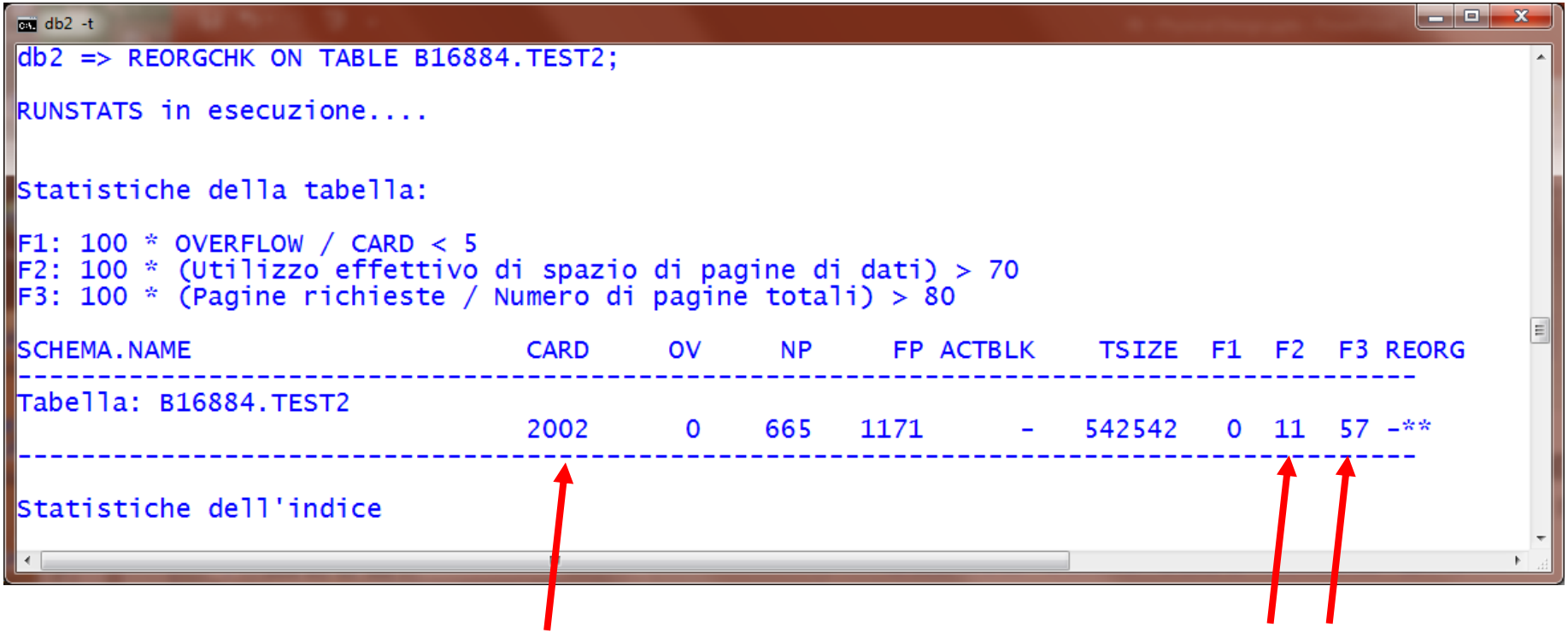
- Le operazioni di riorganizzazione sono ovviamente automatizzabili ed eseguibili specificando diverse politiche

Esempio: frammentare i dati

- Per generare una situazione in cui la riorganizzazione può rendersi necessaria cancelliamo da `TEST2` molte righe:

```
DELETE FROM TEST2 WHERE B>3
```

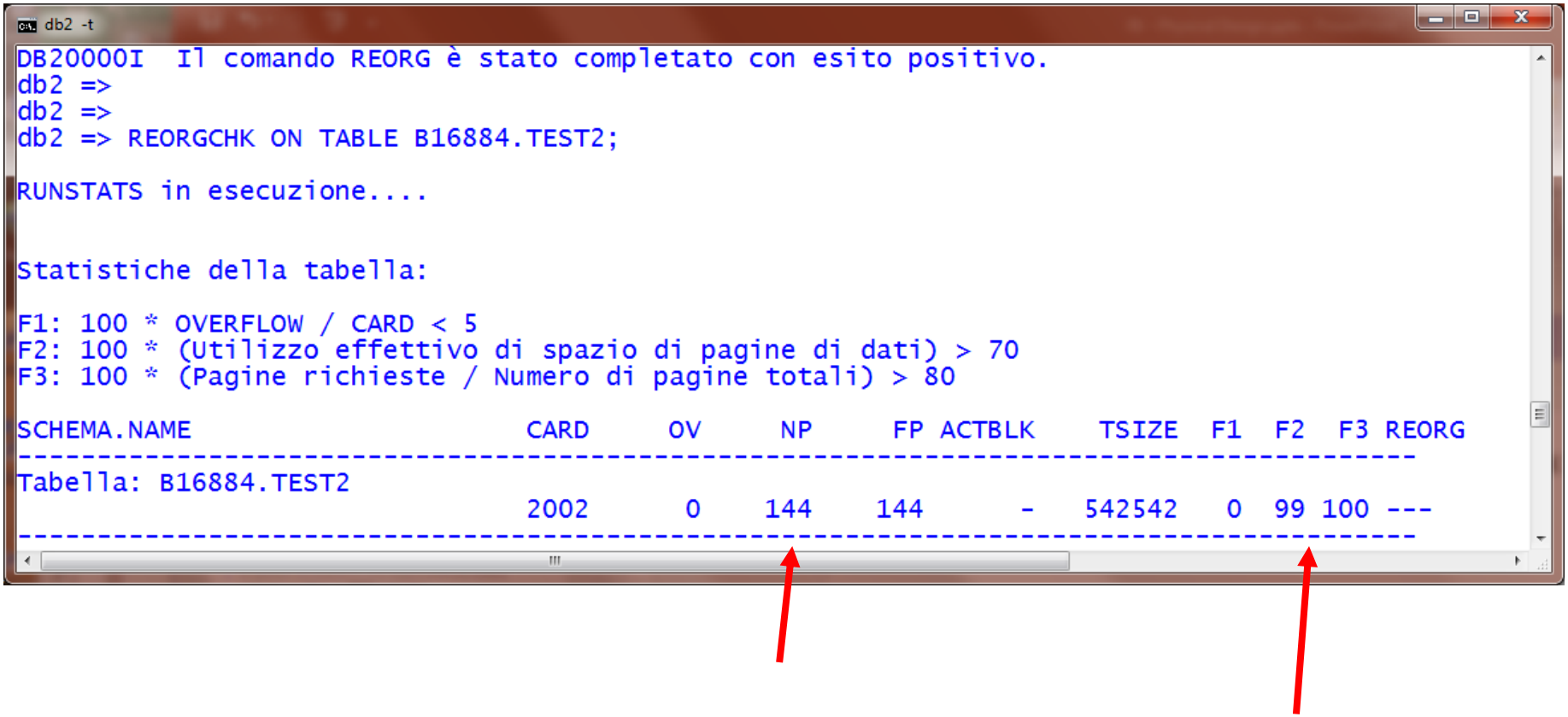
- Rieseguendo `REORGCHK`:



Esempio: riorganizzazione

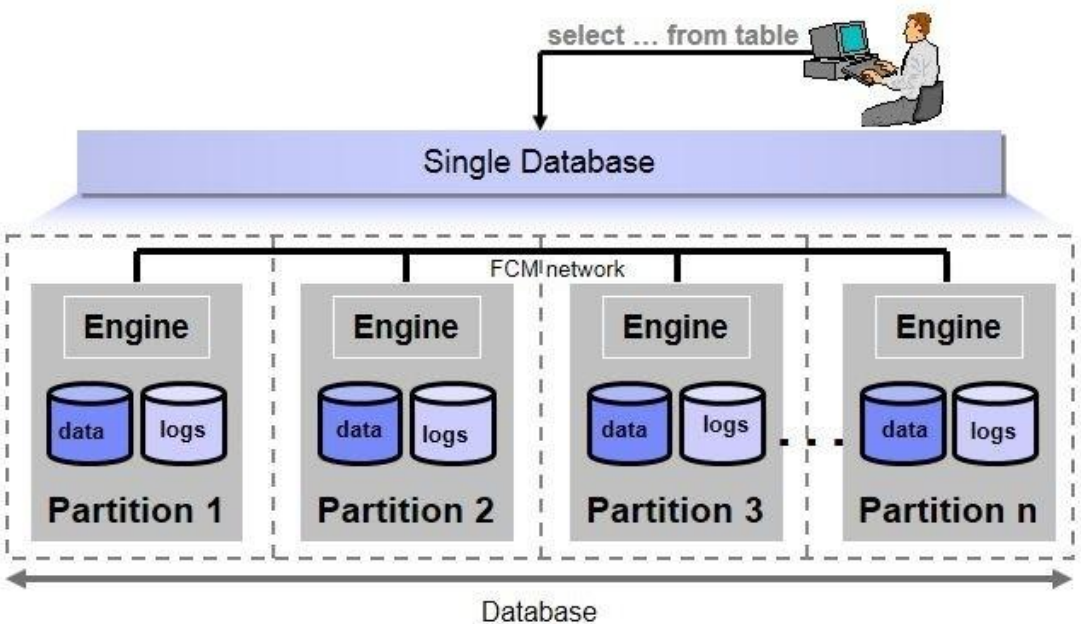
- Si esegue quindi la riorganizzazione fisica:

```
REORG TABLE B16884.TEST2
```



Partizionamento dei dati

- Il partizionamento dei dati ha lo scopo di migliorare le prestazioni:
  - Permettendo di eliminare parti di table in fase di ricerca
  - Permettendo vari livelli di parallelismo nell'esecuzione delle operazioni
  - Distribuendo (e quindi riducendo) il carico di lavoro sui server
  - Riducendo i conflitti tra le transazioni
- A parte tutto ciò che riguarda questioni architetturali (distribuzione, partizionamento dell'intero DB), vediamo le modalità di base con cui è possibile partizionare una table (ed eventualmente anche gli indici)



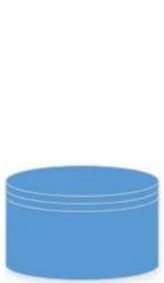
DB partizionato: lato client il DB viene percepito logicamente come se fosse indiviso, ed è il DBMS che si fa carico di suddividere le query sui diversi engine

Partizionamento verticale

- E' il caso in cui gli attributi di una relazione vengono ripartiti su 2 o più table, mantenendo su ognuna la primary key
  - Si ricostruisce l'informazione originale eseguendo (outer) join 1-1
- E' possibile definire viste per fornire una visione integrata della table originale
- Utile in caso di table con molti attributi e/o con attributi BLOB/CLOB (immagini, testo, …)

Key	Product Name	Short Description	Review	Picture
01	Americano @ Starbucks	Black, no sugar	I'd buy again	
02	BB @ Seattle's Best	Black, no sugar	The best	
03	TB @ Zoka Coffee	Black, no sugar	It's okay	
04	BC @ Coffee	Black, no sugar	Never again	

Key	Product Name	Review
01	Americano @ Starbucks	I'd buy again
02	BB @ Seattle's Best	The best
03	TB @ Zoka Coffee	It's okay
04	BC @ Coffee	Never again



Key	Short Description	Picture
01	Black, no sugar	
02	Black, no sugar	
03	Black, no sugar	
04	Black, no sugar	

Partizionamento verticale: esempio

```
SELECT  E.EMPNO,E.FIRSTNME,...,EP.PICTURE,ER.RESUME
FROM    (Employee E LEFT JOIN Emp_Photo EP ON (E.empno = EP.empno))
                    LEFT JOIN Emp_Resume ER ON (E.empno = ER.empno);
```

- In questo caso si usa il `LEFT JOIN` perché non tutti hanno photo e/o CV

EMPNO	FIRSTNME	LASTNAME	WORKDEPT	PICTURE	RESUME
000130	DELORES	QUINTANA	C01	[BLOB]	Delores M. Quint
000140	HEATHER	NICHOLLS	C01	[BLOB]	...
000150	BRUCE	ADAMSON	D11	[BLOB]	...
000190	JAMES	WALKER	D11	[BLOB]	...
000170	MASATOSHI	YOSHIMURA	D11		
000340	JASON	GOUNOT	E21		
000290	JOH				
000110	VINCENZ				
...	..				

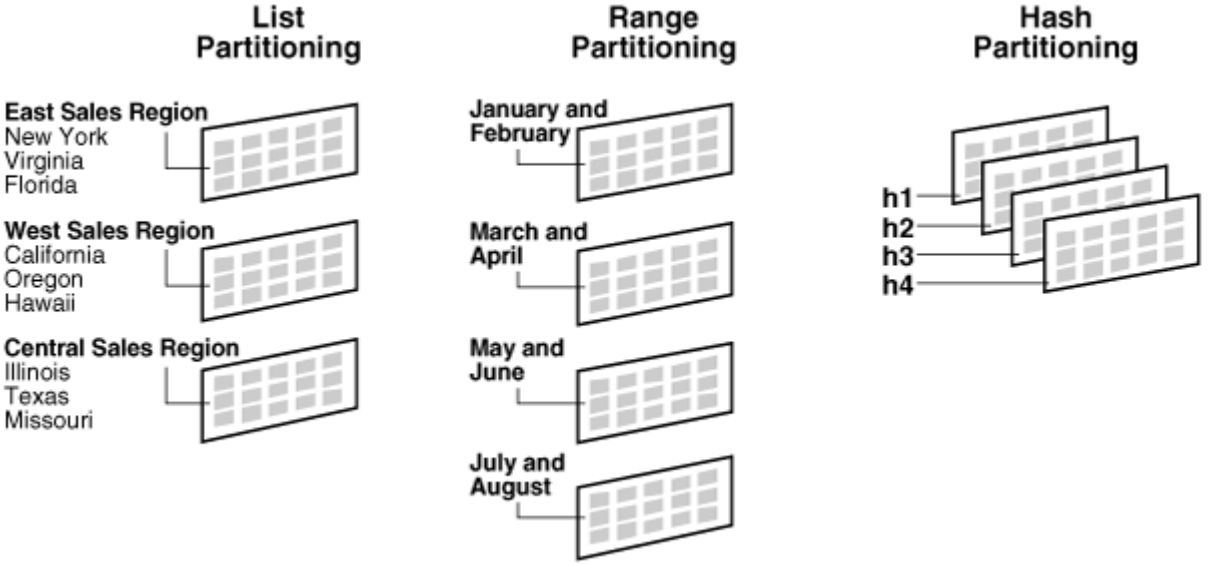
Resume: Delores M. Quintana

Personal Information

Address:	1150 Eglinton Ave Mellonville, Idaho 83725
Phone:	(208) 875-9933
Birthdate:	September 15, 1925
Sex:	Female
Marital Status:	Married
Height:	5'2"
Weight:	120 lbs.

Partizionamento orizzontale

- In una relazione partizionata orizzontalmente sono le tuple che vengono fisicamente ripartite in 2 o più table
- La relazione originale si ottiene facendo la `UNION` delle diverse table
- Permette di superare i limiti sulla dimensione massima di una relazione
- A seconda della tecnica utilizzata si parla di
  - Hash partitioning
  - Range partitioning
    - Oracle ha anche una modalità detta List partitioning, in cui si esplicitano i valori di un attributo che mappano su una specifica partizione



Hash partitioning

- Il partizionamento basato su una funzione hash H ha lo scopo di ottenere una distribuzione uniforme delle tuple sulle diverse partizioni (partizioni più o meno della stessa dimensione)
- Per questo motivo la partitioning key a cui si applica H è tipicamente una chiave, al fine di non dipendere dalla distribuzione dei dati (duplicati)
- Vantaggi:
  - Pruning di partizioni con predicati '=' e `IN` sulla partitioning key
  - Join efficiente tra table partizionate usando la stessa partitioning key
  - Distribuzione uniforme del carico tra diversi dispositivi
- Svantaggi:
  - Interrogazioni di range sulla partitioning key
  - Interrogazioni su attributi non-chiave non usati per il partizionamento
    - Es.: partitioning key: `ORDER_NUMBER`, predicato su `ORDER_DATE`

Range partitioning

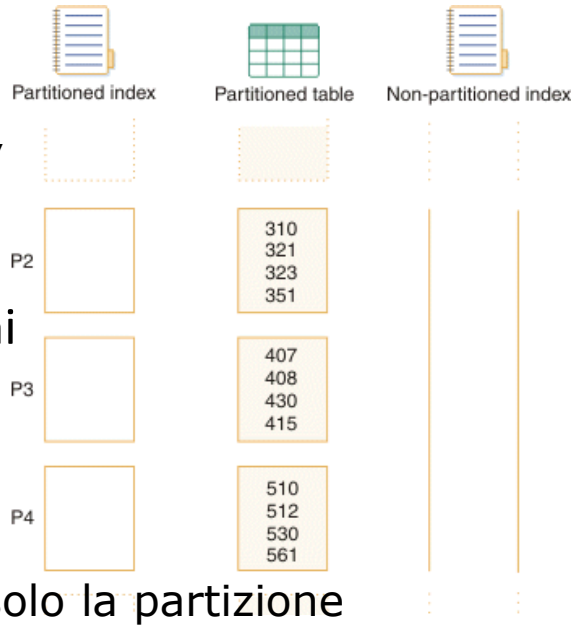
- Il partizionamento basato su intervalli di un attributo (o anche più di uno) ha lo scopo di inserire nella stessa partizione tuple che, a differenza dell'hash partitioning, sono tra loro correlate
- Casi tipici per la partitioning key
  - Attributo temporale (es.: trimestri delle vendite)
  - Attributo spaziale (es.: regioni delle filiali, dei clienti, ecc.)

```
CREATE TABLE PART_R(ID INT PRIMARY KEY NOT NULL,..,DATA DATE NOT NULL)
PARTITION BY RANGE (DATA)
(STARTING MINVALUE ENDING '31.12.2018' IN tbsp1, -- tablespace 1
 STARTING '01.01.2019 ENDING MAXVALUE IN tbsp2); -- tablespace 2
```

- Vantaggi:
  - Pruning di partizioni non coinvolte nella query (predicato di range sulla partitioning key; es.: ordini del 2017)
- Svantaggi:
  - Interrogazioni su attributi non-chiave non usati per il partizionamento

Indici partizionati

- Nel caso di table partizionate è possibile partizionare anche i relativi indici
  - Sia che siano o meno relativi alla partitioning key
- DB2 pone il vincolo che se l'indice è dichiarato `UNIQUE`, allora la chiave dell'indice deve includere la partitioning key per poter essere partizionato
  - Es: se si partiziona su `DATA`, un indice sulla primary key `ID` non può essere partizionato
  - Se si partiziona `ESAMI` su `CODCORSO` e si ha un indice `UNIQUE` su (`MATR`,`CODCORSO`), l'indice può essere partizionato
- Nel primo caso per garantire l'univocità dei valori di chiave bisognerebbe controllare tutte le partizioni dell'indice su `ID`
- Nel secondo caso si controlla solo una partizione dell'indice
  - Es. inserendo (`'00123456'`,`'TBD'`) si controlla solo la partizione corrispondente a `'TBD'`



Partizionamento: response time e throughput

- Il partizionamento di una table può permettere, in funzione della specifica query/transazione:
  1. Il pruning di alcune partizioni
  2. In alternativa, il lavoro da svolgere viene distribuito su più dispositivi e, volendo, anche su più processori
- In entrambi i casi l'effetto netto è, per la query in esame, una riduzione del tempo di risposta (response time), che può esprimersi come la somma di due componenti: Response time = Waiting time + Service time
- D'altronde, nel caso 2, la query impegna più risorse e questo può andare a scapito di altre transazioni, di fatto riducendo il throughput del sistema: Throughput = n. transazioni elaborate/unità di tempo
- E' quindi importante ricordare che non bisogna mirare a migliorare solo le prestazioni per una specifica transazione, ma occorre considerare anche gli effetti che questo può avere sulle altre
- Inoltre va considerato che, per il punto 2, il partizionamento non è necessario, in quanto un DBMS può sfruttare il parallelismo anche su table non partizionate (es. più container in DB2)

Transazioni e SQL server-side

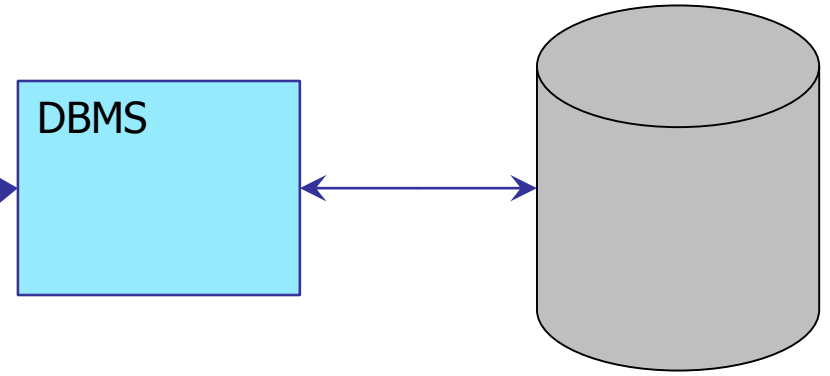
- Tra gli aspetti da considerare per migliorare le prestazioni, uno non secondario riguarda la logica delle transazioni
- In particolare, è opportuno considerare i seguenti punti specifici:
  - Livello di concorrenza: optare per un livello di isolation il più basso possibile, al fine di ridurre le probabilità di conflitti
  - Transazioni complesse: compatibilmente con i vincoli che l'applicazione impone, cercare di evitare transazioni che includono molte istruzioni SQL, in particolare se queste sono inframezzate da input dall'utente
    - Dopo la prima istruzione si attende l'input per la seconda per un tempo indefinito. Se possibile, acquisire prima tutti gli input e poi iniziare la transazione
  - SQL server-side: Spostare lato server quante più operazioni possibili, al fine di ridurre il flusso di comunicazione client-server
- Relativamente all'ultimo punto va ricordato che SQL mette a disposizione due importanti strumenti:
  - Trigger
  - Stored procedures

Trigger

- Senza trigger:

Client

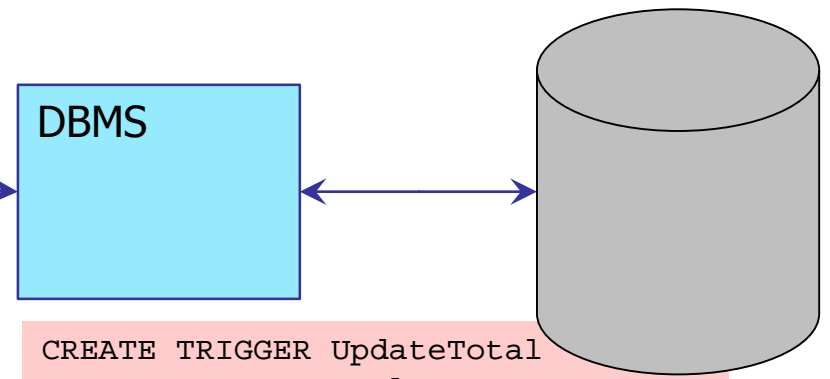
```
INSERT INTO
Sales(Customer,Amount,Date,...)
VALUES('C1234',252.54,'21.10.2019'...);

UPDATE Accounts
SET Total = Total+252.54
WHERE Customer = 'C1234';
```



- Con trigger:

Client

```
INSERT INTO
Sales(Customer,Amount,Date,...)
VALUES('C1234',252.54,'21.10.2019'...);
```




```
CREATE TRIGGER UpdateTotal
AFTER INSERT ON Sales
REFERENCING NEW AS NewSale
FOR EACH ROW
UPDATE Accounts
SET Total = Total+NewSale.Amount
WHERE Customer = NewSale.Customer
```

Stored procedures

- Una stored procedure è una procedura (scritta in SQL o altro linguaggio) precompilata che viene eseguita lato server
- Anche in questo caso è possibile ridurre le interazioni client-server

Stored procedure

```
CREATE PROCEDURE UpdateBonusPoints(IN CUST CHAR(5),
                                   IN TOT DEC(8,2),
                                   IN PTS INT )
LANGUAGE SQL
BEGIN
UPDATE Accounts
SET Total = Total+TOT
WHERE Customer = CUST;
UPDATE BonusPoints
SET Points = Points+PTS
WHERE Customer = CUST;
END
```

Client

```
CALL UpdateBonusPoints('C1234',252.54, 120);
```

Considerazioni finali

- Il problema del design fisico e del tuning di un DB è un problema oltremodo complesso, che non ha nessuna "ricetta" risolutiva
- Esperienza e conoscenza dei principali aspetti che possono influenzare le prestazioni sono i 2 ingredienti di base da considerare
- Vanno inoltre considerate le peculiarità/limitazioni del DBMS specifico, che possono rendere necessarie alcune modifiche alle query, in altri casi non necessarie
  - Esempi: sostituzione delle sub-query con join, riscrittura di predicati con condizioni aritmetiche (E.age=2*D.age anziché E.age/2=D.age), introduzione di predicati ridondanti (R.J=S.J e S.J=T.J implicano R.J=T.J), ecc.
- In generale va sempre ricordato che le scelte fatte non sono mai da considerarsi definitive, ma possono doversi rivedere
  - Ad esempio perché le statistiche sui dati sono cambiate